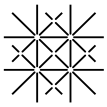

University of Basel

Hamming Width vs Novelty Width and Combinations

Bachelor's Thesis

Faculty of Science of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence
https://ai.dmi.unibas.ch

Examiner: Prof. Dr. Malte Helmert
Supervisor: Simon Dold

Valdrin Sheremetaj
sheremetajv@gmail.com
2021-059-779

03.07.2025

Abstract

The efficiency of classical planning problems varies significantly across problem instances, to explain and potentially exploit these variances, researchers have introduced several width-based complexity measures, such as novelty width and hamming width. These two measures offer different insights to understand the inherent difficulty of planning tasks. In this work, we analyze novelty width and hamming width and compare them, by examining their empirical behaviors and relative strengths. We also explore the implications of combining these width notions, providing a unified framework for understanding their roles in modern AI planners.

Table of Contents

1 Introduction

Planning is an important area in artificial intelligence, which allows a system to decide which sequence of actions it needs to take, to achieve a certain goal. In the context of classical planning, we aim to find such a plan under the assumptions of a static, fully observable environment where all actions are deterministic. Since many general AI planning problems are PSPACE-complete or NP-hard, it makes them computationally infeasible in worst-case scenarios, but still many real-world instances are solved remarkably efficiently by modern planners. Which makes us question: how do we create these effective planners that work well in practice? One promising approach focuses on the so-called width based complexity measures. These measures capture how much of the problem's state space needs to be considered to make progress towards the goal. We will center on two such measures: Hamming Width by Chen and Giménez(2007)[1] and Novelty width by Lipovetzky and Geffner(2012)[3]. They both have different approaches on how to navigate the state space. Hamming Width by Chen and Giménez(2007)[1] focuses on the minimum number of state variables that must be changed in order to reach a new state where more goal conditions are satisfied, without undoing any previously satisfied goals. Novelty width by Lipovetzky and Geffner(2012)[3] on the other hand refers to the minimum number of facts that need to be examined simultaneously in order to determine whether a state is novel—that is, not a duplicate of any previously visited state during breadth-first search. Hence, Novelty width emphasizes the informational value of new states, while Hamming width emphasizes structural locality and incremental improvement. Understanding how these approaches relate to each other is not only theoretically interesting but also practically useful for informing the design of planning algorithms that are both efficient and robust across domains. That's why the overarching goal of this thesis is to conduct a a comprehensive analysis of Hamming Width and Novelty Width—both theoretically and empirically—and to investigate their relationships with each other. We do that by implementing them in the planing system Fast Downward, based on the Search Framework provided by Dold(2021)[2].

2 Preliminaries

In this chapter we introduce important definitions, which we will use throughout this thesis.

2.1 Classical Planning

The classical model for planning is a 5-tuple $\mathcal{S} = \langle S, s_0, S_G, A, f \rangle$, where S a finite set of possible **states**, s_0 is the **initial state**, $S_G \subseteq S$ is the **set of goals**, $a \in A(s)$ denotes the **set of actions**, where $A(s)$ is the set of actions applicable in the state $s \in S$, and f is our **transition function** $f : S \times A \to S$ that maps a state-action pair (s, a) to the resulting state $s' = f(s, a)$. Now we need to clear up some further specifications. We define a **plan** P (for an instance $\mathcal{S}$) as a sequence of actions $a_0, \dots a_m$ that generates a state sequence of $s_0, s_1 \dots s_{m+1}$ such that $a_i \in A(s_i)$, $s_{i+1} = f(a_i, s_i)$ and $s_{m+1} \in S_G$. Intuitively this means that each action a_i is available in the current state s_i, applying a_i to s_i correctly moves us to the next state s_{i+1} and the final state s_{m+1} is a goal state. If a plan from s_0 exists, we call **solvable** otherwise we call s a **dead end**. A **valid** plan is a path through, the state space, starting at s_0, following applicable actions that takes us from one state to the next, ending in a state that meets a goal. We call a problem instance $\mathcal{S}$ **tractable** if there exists an algorithm that, given $\mathcal{S}$, that computes a valid plan $P = \langle a_0, \dots, a_m \rangle$, or determines that no such plan exists, in polynomial time with respect to the given input.
FORMALIZATIONS OR GENERAL DEFINITION?

2.2 State Spaces?

We will use State Spaces to give us a more structured way to represent all the possible configurations or situations that a system could be in.
A state space is a 6-tuple $\mathcal{S} = \langle S, A, cost, T, s_I, s_G \rangle$ with a finite set of possible **states** S, finite set of **actions** A, **action costs** $cost : A \to \mathbb{R}_{\checkmark}^{+}$, the **transition relation** $T \subseteq S \times A \times S$, an **initial state** $s_I \in S$ and a **goal state** or set of goal states $s_G \subseteq S$. We will visualize these state spaces as directed and labeled graphs, where nodes represent our states and edges represent our actions, labeled with the actions cost of the corresponding transition.
MATHEMATICAL FORMALIZATION REQUIRED IF NEEDED IN THESIS

2.3 Mathematical Tools for Width Measures

To define and analyze the different notions of width in planning problems, as introduced later in this thesis, we require a number of basic mathematical constructs. These form the foundation for formally expressing plan execution, measuring distances between states and characterizing novelty in search. While the actual definitions of width measures will follow in their dedicated chapters, this section introduces the formal elements they depend on.

2.3.1 Plan Execution Notation

Let $P = \langle a_1, a_2, \ldots, a_n \rangle$ be a plan, and let s be a state. Then we define the application of P to s, written $s[P]$, as the result of applying all actions in sequence:

$$s[P] := s_n \quad \text{where} \quad s_{i+1} = f(s_i, a_i),\ s_0 = s.$$

This notation will be used to express the outcome of applying a plan to a given state.

2.3.2 Hamming Distance

Given two states s_1 and s_2 over the same variable set V, the *Hamming distance* between s_1 and s_2 is defined as:

$$d_H(s_1, s_2) := |\{v \in V \mid s_1(v) \neq s_2(v)\}|\,.$$

It measures the number of variables whose values differ between the two states.

2.4 Search Framework

We will introduce the **Search Framework** by Dold(2021), which we will use for our implementation of the Hamming Width Search and Novelty Width Search algorithms. It allows us to specify search algorithms, by only 3 subroutines named **progressCheck**, **expandCheck** and **updateClosed**. We will implement these algorithms ourselves according to the properties of our Hamming- and Novelty Width Search. The pseudocode for the algorithm provided already by Dold(2021) is shown here.
It defines a general skeleton for our search algorithms in planning tasks, by searching through states, while trying to reach a goal state γ and managing both open (states to explore) and closed states (states already explored). It does that by initializing first, while checking if the goal γ is already reached. If it does agree the search is already over. If it is not reached already, we initialize the open list containing only the initial state I. The closed set is initialized with the empty set, symbolizing that no state is reached yet and immediately updated by the subroutine **updateClose** with the initial state I. Dold(2021) also added a reference state *reference*, which first remembers the starting state and later helps to see if progress was made. After the initialization, we enter the main while loop, which runs as long as the open list contains at least one state. Inside the loop we first assign the first element of the open list to the current state *current*, then we iterate (in an arbitrary order) over each *candidate* that are a successor of current with a for-loop. For each of those *candidates*, we will first check if we already found a goal, we then return the path and the search is over. If

Algorithm 1 Search Framework

Data: planning task $\Pi = \langle V, I, O, \gamma \rangle$, width $k \in \mathbb{N}$, heuristic h,
subroutines `updateClosed`, `progressCheck`, `expandCheck`
Result: plan π
if $\gamma \subseteq I$ **then**
    return empty plan
end if
$open := [I]$
$closed := \emptyset$
`updateClosed`$(I, closed, k)$
$reference := I$
while *open* is not empty **do**
    $current :=$ pop first element of *open*
    for all $candidate \in succ(current)$ **do**
        if $\gamma \subseteq candidate$ **then**
            return extracted path to *candidate*
        else if `progressCheck`$(candidate, reference, h)$ **then**
            $open := [candidate]$
            $closed := \emptyset$
            `updateClosed`$(candidate, closed, k)$
            $reference := candidate$
            break
        else if `expandCheck`$(candidate, closed, k)$ **then**
            `updateClosed`$(candidate, closed, k)$
            append *candidate* to *open*
        end if
    end for
end while
return fail

this is not the case, we check if progress was made with the **progressCheck** subroutine. If that's true, we reinitialize the open list, the closed set and the *reference* state. The closed set becomes the empty set again, and will be updated again by **updateClosed** while using the *candidate* state. The open list then contains only the *candidate* state and *reference* gets updated to *candidate* and the for loop breaks. If candidate both provides no progress and is not agreeing with the goal, then we test *candidate* if it should be expanded further. We accomplish this, by checking it with the subroutine **expandCheck**. If *candidate* should be expanded further, we call the subroutine **updateClosed** on *candidate* to add elements into the closed set and append *candidate* to the open list. If the while loop ends, no solution was found and it terminates with a fail return.

3 Width-Based Complexity Measures

3.1 General Idea of Width in Planning

In automated planning, *width* is an important notion which characterizes the inherent difficulty of a planning problem. Intuitively, width shows us how many subgoals or facts have to be handled together at any time to find a solution. Many planning problems, can be solved by focusing on a small number of relevant facts or subgoals at once, which makes the planning problem often much easier to solve. This means that there is a number (the width) which is bounded by a small constant. In fact, if the width of a planning instance is bounded by a constant, the search for a plan needs time exponential only in that constant (and polynomial in the size of the instance), making such problems tractable in practice. As mentioned before, we will focus on two different formal notions of width. Below we will explain each notion in turn and then compare how they complement each other.

3.2 Hamming Width

Chen and Giménez(2007) [1] introduced a width measure for classical planning problems that is centered on achieving goals while limiting interference between subgoals. The key idea is to measure how many state variables might need to be changed in order to achieve another subgoal. A planning instance $\mathcal{S} = \langle S, s_0, S_G, A, f \rangle$ is said to have width k if, informally, from any state, one can achieve any single unsatisfied goal variable by a plan that "changes no more than k other variables". In other words, to make progress on any subgoal, you never need to disturb more than k of the other goals. If k is small, subgoals can be handled mostly in isolation, implying a low overall complexity. We outline the formal definitions of the width measures introduced by Chen and Giménez now:

Definition 3.1 (Width by Chen and Giménez(2007) [1]). Let $\mathcal{S} = \langle S, s_0, S_G, A, f \rangle$ be a planning instance, and let s be a reachable state

3.3 Novelty Width

3.4 Conceptual Comparison

4 Theoretical Analysis

4.1 Formal Properties of Hamming Width

4.2 Formal Properties of Novelty Width

4.3 Relationship

5 Implementation

5.1 Implementation of Hamming Width Search

5.2 Implementation of Novelty Width Search

6 Empirical Evaluation

7 Conclusion

Bibliography

[1] Hubie Chen and Omer Giménez. Act local, think global: Width notions for tractabla planning. In *ICAPS*, 2007.

[2] Simon Dold. Correlation complexity and different notions of width. Master's thesis, University of Basel, 2021.

[3] Nir Lipovetzky and Hector Geffner. Width and serialization of classical planning problems. In *ECAI*, 2012.

A Appendix

University of Basel

Faculty of Science

Declaration on Scientific Integrity

(including a Declaration on Plagiarism and Fraud)
Translation from German original

Title of Thesis:

Name Assessor: _____

Name Student: _____

Matriculation No.: _____

I attest with my signature that I have written this work independently and without outside help. I also attest that the information concerning the sources used in this work is true and complete in every respect. All sources that have been quoted or paraphrased have been marked accordingly.

Additionally, I affirm that any text passages written with the help of AI-supported technology are marked as such, including a reference to the AI-supported program used. This paper may be checked for plagiarism and use of AI-supported technology using the appropriate software. I understand that unethical conduct may lead to a grade of 1 or "fail" or expulsion from the study program.

Place, Date: _____ Student: _____

Will this work, or parts of it, be published?

No

Yes. With my signature I confirm that I agree to a publication of the work (print/digital) in the library, on the research database of the University of Basel and/or on the document server of the department. Likewise, I agree to the bibliographic reference in the catalog SLSP (Swiss Library Service Platform). (cross out as applicable)

Publication as of: _____

Place, Date: _____ Student: _____

Place, Date: _____ Assessor: _____

Please enclose a completed and signed copy of this declaration in your Bachelor's or Master's thesis.